Exploring Process Groups for Reliability, Availability and Serviceability of Terascale Computing Systems

Daniel I. Okunbor[1]
Professor, Department of Mathematics and Computer Science Fayetteville State University 1200 Murchison Road Fayetteville, NC 28301
Christian Engelmann and Stephen L. Scott[2]
Research Scientists, Computer Science and Mathematics Division Oak Ridge National Laboratory Oak Ridge, TN 37831,

Abstract

This paper presents various aspects of reliability, availability and serviceability (RAS) systems as they relate to group communication service, including reliable and total order multicast/broadcast, virtual synchrony, and failure detection. While the issue of availability, particularly high availability using replication-based architectures has recently received upsurge research interests, much still have to be done in understanding the basic underlying concepts for achieving RAS systems, especially in high-end and high performance computing (HPC) communities. Various attributes of group communication service and the prototype of symmetric active replication following ideas utilized in the Newtop protocol will be discussed. We explore the application of group communication service for RAS HPC, laying the groundwork for its integrated model.

1 Introduction

Recent developments in terascale computing have been aided by cluster computing, which is a distributed computing environment consisting of loosely coupled computers working together to solve one "large" problem [45]. Computer clusters are regarded as commodity clusters since they are generally made of low-cost commercial-off-the-shelf (COTS) computers, connected through fast local area networks. The concept of computer clustering is not new, the first computer cluster was built in the 1950s and many cluster-based computer systems have been developed ever since [45]. There has been a tremendous growth in research interests in the application of computer clusters in HPC within the last two decades, see [38] and [45]. This is due largely to the rapid improvements and dramatic cost reduction in computing technologies in both hardware and software. The competitive price/performance ratio of commodity clusters poses a serious challenge for traditional massively-parallel-processors supercomputers [3], [22], [24]. The em-phases in HPC have shifted to commodity clusters, for reasons of easy deployment, interoperability, flexibility, scalability, upgradeability and low-cost [23], [26]. Message passing libraries, such as the Parallel Virtual Machine (PVM) and Message Passaging Interface

[1] This research was supported in part by an appointment to the Oak Ridge National Laboratory/Oak Ridge Associated Universities Historically Black Colleges and Universities and Minority Education Institutions Summer Faculty Research Program.

[2] Supported in part by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

(MPI), providing interprocessor communication amongst computing nodes, propelled research in parallel computing adopting computer clusters [38], [45]. They are considered the de facto standards for programming parallel systems. These libraries and many of their variants (OpenMPI, Local Area Multicomputer MPI, MPICH, Harness) and several new others have been developed over the years to meet different needs in HPC [3], [4]-[15]. High performance computing has also benefited in great measures from advances in algorithms for distributed systems. Better job schedulers, such as the Portable Batch System (PBS) (and subsequently, OpenPBS), Condor, Sun Grid Engine (SGE), and Maui have enabled development of several HPC clusters [4]-[15].

The proliferation of software systems supporting cluster computing led to many variations of software techniques for parallel computing. Several of the parallel processing software systems are application-specific, limiting their wide utilization. To mitigate this complexity of cluster configuration, several cluster management systems, such as the Open Source Cluster Application Resource (OSCAR) and Rocks have been developed, see [45] for detailed information about both software packages.

The architecture of a computer clusters comes in two flavors, namely, symmetric and asymmetric [45]. In a symmetric model, the computing nodes are connected to an existing communication network and they function as individual computers, accepting requests to users directly from the external network and Internet; the asymmetric model has a node designated as the headnode to which all users' requests are sent [45]. The headnode or the frontend serves as gateway for the rest of the compute nodes. These configurations have associated with them issues such as reliability, availability and serviceability that are pertinent to high-end, high performance computing. The concepts of reliability, availability and serviceability are interrelated and connote different things to different people [47], [48]; to the hardware manufacturers, RAS means that hardware system works continuously even in the presence of failure in any of the constituent hardware components; to software developers, RAS indicates failures in the underlying processes and hardware components do not cause any interruptions in the overall system performance. A computer system that satisfies RAS criteria are fault-tolerant - in a hardware realm, an RAS system implies building a fault-tolerant system that possesses the capabilities of sustaining failures.

This paper describes various aspects of reliability, availability and serviceability in high performance computing. While the symmetric model plausibly supports the high available cluster architecture, it is difficult to manage due to problems associated with workload distribution and reduced optimal performance [45], [50]. As indicated above, the headnode in the asymmetric model serves as the primary server, providing public interface to the entire cluster and hence, constitutes a single point of failure (SPOF), and therefore, a degradation for RAS [4]-[15]. The asymmetric model has been commonly utilized in many cluster systems due to it ease in implementation and coordination of interconnected computing nodes. The cluster management systems, OSCAR and Rocks implement the asymmetric model, which means that while both toolkits may be highly valuable in high performance computing, they suffer from the single point of failure and hence, may not have high availability property.

There is a growing number of research interests in developing RAS software systems embedded in the cluster management systems. Examples are the highly available Linux system (HA-Linux), highly available OSCAR toolkit (HA-OSCAR), and fault-tolerance MPI (FT-MPI) [3], [24]-[26]. This paper would focus on algorithms for developing RAS, fault-tolerant cluster systems. Distributed algorithms for process group systems (group communication services), reliable multicast/broadcast, total order (atomic) multicast/broadcast, virtual synchrony have consistently been described in the literature to support RAS and fault-tolerant systems. This paper will discuss algorithms for process groups in the context of RAS for high performance computing. Algorithms for process groups are characterized using safety and liveness properties. Many variations of these classifications and specifications are prevalent in literature. A particularly interesting categorization was one given by Morgan and Ezilchelvan [43] and implemented in the Newcastle Total Order Protocol (Newtop). This would be presented in this paper in addition to description of broadcast service developed and implemented by Oestreicher [49]. These specifications would pave the way for the design, development and implementation RAS for terascale computing.

2 High Availability

Developing highly available HPC systems and thereby, increasing the quality of HPC software and networks, is a critical component in the US legislation entitled, “The High-Performance Computing Revitalization Act.” The bill establishes security standards and practices for critical systems serving critical infrastructures, the national defense, and advanced scientific applications [51]. To understand the concept of high availability, it is important to describe the concepts of reliability, availability and serviceability. RAS are essential attributes of a computing system design [49], [50]. A reliable system is one that consistently produces the same results and thus, maintaining data integrity. Reliability pertains to system’s ability to operate continuously without failures or outages. An outage or fault refers to deviation from the specified behavior of the system. Faults or outages may be due to error in design, failure in hardware component, over-loading of engineered system resources, and error in executing procedures [47]. Mean Time Between Failure (MTBF), Failures-In-Time (FITS), Mean Time To Failure (MTTR), Mean Time to Repair (MTTR) are parameters generally used for measuring system reliability. Availability is the degree to which a system suffers degradation or interruption in its service to the customer as a consequence of failures of one or more of its parts. It is the percentage of time when a system is operational. Resnick [47] distinguished basic availability from continuous and high availability. A system is said to have basic availability if it is designed, implemented and deployed with sufficient components (hardware, software and procedures) to satisfy the system’s functional requirements; high availability is designed, implemented and deployed with sufficient components to satisfy the system’s functional requirements and also has sufficient redundancy in components (hardware, software and procedures) to mask certain defined faults or outages [47]. While high availability masks unplanned outages, continuous availability masks both planned and unplanned outages. Using the reliability parameters above, availability can be expressed mathematically as

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Note that MTBF = MTTF + MTTR [47]. The computed availability is typically specified in terms of nines. A 1-nine is 90% availability, 2-nines is 99% availability, 3-nines is 99.9% availability, and so on. The downtime of 1-nine available system is 36.5 days per year; 2-nines available system is 3.65 days per year. Apparently, the higher the number of nines the highly available the system becomes. The importance placed on high availability continues to resonant in high performance computing. High availability attribute in hardware and software systems has repeatedly been modeled using redundancy or replications. This led to a further characterization of computer clusters, namely, high availability clusters, load balancing clusters and high performance clusters. High availability cluster systems are implemented for the purpose of improving the availability of services that the system provides [3], [45], [47], [50]. Examples of software systems supporting high availability clustering are the HA-Linux, HA-OSCAR, Distributed Replicated Block Device (DRBD), and FT-MPI. Load balancing clusters use one or more load-balancing front ends to distribute workload to a collection of back end servers. The Beowulf-based cluster systems using OSCAR or Rocks are examples of load balancing clusters. The high performance clusters provide increased performance by splitting a computational task across many different nodes in the cluster. The distinction between all three categories is very subtle, high performance and load balancing are characteristics of high availability cluster systems. High availability is crucial in high performance computing. The last of the RAS concept is serviceability and it is defined as the ease with which corrective maintenance or preventative maintenance can be performed on a system. Clearly, serviceability improves system availability and reliability.

Redundancy or replication, group communication and virtual synchrony play central role in the design and implementation of high availability clusters. Redundancy or replication would be discussed in this section and group communication and virtual synchrony, in the following sections.

There are three levels of replications that are essential in the description of high availability -cold standby, warm standby and hot standby. The cold and warm standby are also called passive replication and hot standby, active replication. In the case of cold standby, the redundant (or secondary) component serves as a backup and it is only activated whenever the primary component fails. The secondary system receives scheduled backups and it is initialized to enter service to accept rollback to the last consistent and committed state of the failed server. The warm standby model is similar to cold standby, except that the scheduled backups are done more frequently, mirroring the primary system at regular intervals. In the hot standby, the redundant (or backup) server runs concurrently with the primary component, mirroring the primary server in real time so that both servers contain identical information. Many software systems supporting high availability and implementing passive and active replication model or their combinations have been developed recently. Majority of these HA systems run on the open source Linux operating systems and many have claimed success in mission-critical applications. Existing HA systems (HA-OSCAR, Linux-HA, and ShaoLin HA Cluster) support two server nodes, one primary and the other cold or warm or hot standby server node [4]-[15].

3 Group Communication Service

Distributed systems are commonly described abstractly as consisting of the application layer at the top level and communication network layer at the bottom layer and the centralized level or middleware, made of the group communication layer. The centralized group communication layer simulates the group communication service (GCS) and it is critical in the design of distributed systems. The group communication service, sometimes called view-oriented group communication service, has been an active research in distributed computing for more than two decades. The terms group membership service and process groups have been used to describe group communication. Without loss of generality, we would use group communication service in this paper. Distributed computing devises algorithms for a set of processes that seek to achieve some form of cooperation [2], [1], [16]-[21], [29]-[44]. The distributed system may permit some of the processes to fail, leave or join; reconstituting a new set of processes for the group. This complicates algorithms supporting group communication service. GCS provides two key services, namely, a membership service and a multicast service. The multicast service is a means by with the processes in the group communicate with each others, while the membership service provides a tracking mechanism for processes currently in the group (the view) [42], [44], [46]. Depending on the application, the multicast service may be replaced with broadcast service, in which case, all processes in the group are expected to receive the same requests. In a multicast service, requests are received by a selected number of processes in the group. The terms primary partition and partitionable group membership service are used to distinguish group communication services allowing single view of a group (broadcast) and multiple views of the group (multicast), respectively. However, it must be pointed out that the distinction between broadcast and multicast have not be made clear. The view of a group is the collection of the current participating processes within the group. A dynamic process group is one in which group membership changes by allowing processes to leave, join or be removed due to failures. A primary partition process group designates a process group partition as the primary partition, whose member processes are allowed to deliver messages. On the contrary, a partitionable process group allows all processes to deliver messages regardless of the partition they belong [40]-[44].

The concurrency in the operations of the processes in GCS makes it an excellent candidate for replication, which, as indicated in the preceding section, supports high availability, making GCS a difficult problem to tackle. It is not surprising that there are approximately sixty-four group communication algorithms reported in literature. Additional motivation for the support of group communication service for high availability systems stems from its ability to integrate failure detection algorithms (hence, allowing for the construction of fault-tolerant dynamics necessary for RAS), virtual synchrony (the topic for next section), and a variety of other interesting semantics. GCS applications include state machine or active replication; primary backup replication; distributed and clustered operating systems; distributed transactions and database replication; resource allocation; load balancing; system management and monitoring; and high available servers. This section describes the specifications (also called semantics) and implementation of group communication. The basic configuration of group communication service will be presented as well as the safety and liveness properties required for the characterization of different group communication protocols or primitives. We first provide definitions for basic group communication terminologies [2], [1].

We use the abstraction provided in [2], [1], [41], [42] to describe group communication service. A group communication service is assumed to compose statically of a set of N processes, uniquely identified as p1, p2,..., pN. Processes communicate by exchanging uniquely specified messages through communication links. A correct process is one that does not exhibit faulty behaviors resulting from crash, omission, timing or Byzantine failures; it must successfully executes all assigned automaton, a sequence of local and global events defining the algorithm of the process, either within a bounded unit of time (synchronous algorithm) or an unbounded time units (asynchronous algorithm). An event may involve one process receiving a message from another process, executing a local computation or sending a message from one process to another. The performance of a distributed algorithm is measured by the latency, the amount of time for a message to be delivered from one process to the other, bandwidth of the communication links, and throughput, the speed at which local events are performed. Performance analysis of the distributed algorithm(s) described here would be presented in another report, however, it is worth mentioning that while some researchers have claimed successes in comparative study using performance analysis for different group communication services, much still need to be done in research terms in this area. In the remainder of this section we would discuss the reliable order and total order broadcast algorithms; safety and liveness properties; mechanisms for message ordering in group communication systems.

In describing the safety and the liveness properties, we assume a static single process group view and for simplicity, the events are represented by primitives, namely, broadcast(m) (this is assumed to be executed once) and deliver(m),where m is the message. As indicated above, a message must be uniquely defined by its original sender, (in this case, we define an operation sender(m) to return the sender process identifier), the sequence number assigned by the original sender, local lock together with the process identifier [2], [1], [44].

Definition Process p **broadcasts** message m if p executes broadcast(m).

Definition Process p **delivers** message m if p executes deliver(m).

Using these two definitions and sender(m) we state the safety and liveness properties [2].

Validity: A correct process that **broadcasts** a message m eventually **delivers** it.

Uniform Agreement: If a process p **delivers** a message m, then every correct processes must **delivers** m.

Uniform Integrity: Every process **delivers** message m at most once only if m was previously **broadcasts** by sender(m).

Uniform Total Order: If process p broadcasts messages m1 and m2, and q broadcasts m1 and m2, then p **delivers** m1 before m2, if and only if q **delivers** m1 before m2.

The safety properties include the Uniform Integrity and Uniform Total Order properties, which when violated at time t would never be satisfied. Validity and Uniform Agreement properties form the liveliness properties, which mean that these properties are eventually guaranteed to hold at some time t. A reliable broadcast group communication service is one satisfying Validity, Uniform Agreement, Uniform Integrity properties and a total order broadcast or atomic broadcast group communication service satisfies all four properties. Let dst(m) be the set of processes at which m is delivered. We define total order broadcast and multicast as

Total Order Broadcast: dst(m)= {p1,p2,…,pN} for all m.

Total Order Multicast There exists a message m for which sender(m) is not equal to dst(m) and messages mi and mj for which dst(mi) is not equal to dst(mj).

There are non-uniform counterparts of Uniform Agreement and Uniform Total Order properties that permit only correct processes to deliver messages, weakening the restriction placed on faulty processes. To design fault-tolerant total order broadcast may require both uniform and non-uniform properties. While we would not discuss non-uniform properties any further, it must be remarked that these properties may be beneficial in the design of RAS systems using group communication service. Safety and liveness properties do not guarantee the absence of contamination in correct processes, since it is possible for a faulty process to reach an inconsistent state before crashing [2], [1]. Ordering properties are used to ensure that processes deliver messages that may not lead to an inconsistent state. Let tr(p) be the trace of process p, that is, the set of all events at p, the following are the ordering properties.

Prefix Order: For any two processes p and q, either tr(p) prefix tr(q) or tr(q) prefix tr(p).

Gap-Free Uniform Total Order: If some process **delivers** message m1 after message m2, then a process **delivers** m1 only after it has delivered m2.

FIFO Order: If a correct process **broadcasts** m before m, then no correct process **delivers** m before m.

Causal Order: If the broadcast of m1 causally precedes m2, then no correct process **delivers** m1 before m2.

Local Order: If a process **broadcasts** m1 and a process **delivers** m1 before it **broadcasts** m2, then no correct process **delivers** m2 before m1.

In a total order broadcast and multicast protocols, the algorithms or primitives for ordering messages are very important. As mentioned earlier, a message is identified by its sender, destination or sequencer process. Depending on the process group dynamics, the sender and destination processes are easily determined. The sequencer process assigns a unique sequence number to the message. The order in which messages are delivered is a crucial problem in group communication service. While there are many message ordering mechanisms, we focus on a few discussed in [41], [42] - fixed sequencer: a single process is elected to serve as the sequencer for all messages, there are three variants of this (unicast to broadcast, broadcast to broadcast, unicast to unicast to broadcast); moving sequencer or token-based: amongst the possible sender processes, one is elected to serve as a sequencer for each message using token, where a token passes from sender process to next, token-ring approach has prominently been utilized for moving sequencer; privileged-based: in this case, the sender process broadcast messages only when granted the privilege using the process of arbitration; communication history: uses timestamps in message delivery while ensuring total order, two variants of this algorithm are causal history and deterministic merge; and destinations agreement: delivery order is based on agreement amongst destination processes. Existing algorithms, Ameoba developed by Kaashock and Tanenbaum, MTP by Armstrong et al., Tandem by Carr, Isis by Birmann et al., Phoenix by Wilhelm and Schiper, Rampert by Reiter, use the fixed sequencer algorithm; RMP by Whetten et al., DTP by Kim and Kim, Pinwheel by Cristian et al., support the moving sequencer; On-Demand by Cristian et al., Train by Cristian, Token-FD by Ekwell et al., Totem by Amir et al., TPM by Rajagopalan and McKinley, RTCAST by Abdelzaher et al., MARS by Kopetz et al., are privileged-based algorithms; Lamport by Lamport, Psync by Peterson et al., Newtop by Ezilchelvan et al., Trans-Total by Moser et al., ATOP by Chockler et al. CORel by Keida and Dolev, Deterministic Merge by Aguilera and Strom, HAS by Cristian et al., are based on communication history; Skeen by Birman and Joseph, Prefix Agreement by Anceaume, Quick-A by Berman and Bharali are destination agreement algorithms.

We end this section by discussing in depth the New-top protocol. In their motivation for Newtop (acronym for Newcastle Total Order Protocol), Morgan and Ezilchelvan [43] presented an interesting characterization for replication, which, as previously mentioned, offers high availability in the presence of failures. They distinguished between request dissemination (D) and reply collection (C), which are two guiding principles of how messages are sent, delivered and replies collected by replicas. D1 represents a situation in which a request is sent directly to one replica called request manager and D2, when the request is broadcast to the entire group. C0 denotes the case when the sender waits for no reply; C1, the sender waits for one reply; C2, the sender waits for all replies from destination processes; C3, the sender waits for replies from a majority of the destination processes. R1 is used to represent passive replication: a designated replica server called primary executes the sender process' request and multicasts to other replicas, note that, multiple process groups are acceptable; R2, for active replication: all replica servers execute sender process' request. O1 is used to denote asymmetric ordering: a replica is designated to assume the responsibility of ordering the messages; O2, to denote symmetric ordering: in this case, all members use the same deterministic algorithm for message ordering. A group communication policy is formed by a combination of attributes D, C, R and O. Clearly, R1 cannot combine with C2 or C3, giving a total 24 different policies. The Newtop protocol is based on partitionable process group that allows different asymmetric and symmetric message orderings in different groups. That is, message ordering may be symmetric in one process group and asymmetric in another. The protocol supports a variety of group communication policies. A policy including R2 has been successfully argued to support high availability. It would be interesting to see performance analyzes of several of the group communication policies for high performance computing applications in multiple process group situations.

4 Virtual Synchrony

Group communication services supporting virtual synchrony are guaranteed to have identical set of messages delivered in the previous view of the process group. In a virtual synchrony, membership change and view are very important. In this section, we would discuss the various concepts of virtual synchrony and how there are used to enhance group communication service. A variation of virtual synchrony, called extended virtual synchrony would also be described. What is the hype about virtual synchrony and extended virtual synchrony? Why has virtual synchrony been so important in the development of group communication service? Why it is necessary to enforce virtual synchrony in the design of RAS for high performance computing? All these questions and many more would be addressed in this section.

A group communication service system that has a virtual synchrony communications (VSC) [41] must have in addition to the set of events or primitives for group communication service describe the following: the viewchng which returns new view of the process group consisting of the current participating processes while executing an event; send(m) and receive(m) for point-to-point broadcast communications. Virtual synchrony properties are

If a correct process p sends a message m to a correct process q, then q eventually receives m.

For each message m, a process p receives m at most once, and only if m was previously sent to p by some process q.

If a correct process p **broadcasts** a message m, then it eventually **delivers** m.

For each message m, each process **delivers** m at most once, and only if m was broadcasted by some process.

If a process (respectively, a correct process) p **delivers** m in view v, then all processes which are either correct or faulty deliver in a view change event v delivered m.

If a process p **delivers** a message m in view v1 and a process q **broadcasts** m in view v2, then v1 = v2.

We have already seen that the first four properties are satisfied by total order broadcast. The last two properties enforce virtual synchrony in a group communication service, guaranteeing that messages are delivered in the view they were sent and hence, supporting fault-tolerance necessary for building RAS systems. In virtual synchrony, processes installing new view based on the previous view must have the same messages received in the old view. Therefore, virtual synchrony is essential for applications implementing replication using state machine approach (active replication). According to Chockler, Keidar and Vitenberg [1], a group communication system that supports virtual synchrony allows processes to avoid state transfer among processes that “continue together” from one view to another. The extended virtual synchrony introduces transitional set, agreement on successors and safe message properties to virtual synchrony. A transitional set contains information that allows processes to locally determine whether the virtual synchrony applies or a state transfer is required to create transitional view. This is implemented in Tansis and Totem by Moser et al. The agreement successor property ensures that every member process in the intersection of p’s current view and p’s previous view is also coming from the same view [3]. It is implemented in Horus by Friedman and Vaysburg and Ensemble by Hickey et al. The safe message property, implemented in Transis and Totem, guarantees that a process p receives a message m if the group communication service knows that the message is stable, that is, all member processes in the current view have received the message from the network.

One can not overestimate the relevance of group communication service that supports virtual synchrony and extended virtual synchrony in RAS systems for high performance computing. Symmetric active replication group communication service for high availability clusters will benefit from consistency in message delivery achievable in virtual synchrony and extended virtual synchrony group communication service.

5 Fault-Tolerance

Building a group communication service that is fault-tolerant requires mechanisms for detecting process and message delivery failures and for taking corrective and recovery measures for the failed processes. Synchronous and asynchronous group communication systems are categorized using the parameters, the process speed interval, the difference between the speeds of the slowest and fastest process; and communication delay or communication latency [2]. While a synchronous system places a bound on the parameters, for asynchronous systems, these parameters are unbounded. By making values of these parameters to be infinitesimally small, it is possible to have asynchronous systems, called timed asynchronous systems. Using oracles that processes can query, fault-tolerant systems are constructed implementing timed asynchronous systems. Failure detectors and randomized values are examples of oracle. Failure detectors are classified into perfect, eventually perfect, strong, and eventually strong. All classes satisfy the property that every faulty process is permanently suspected by all correct processes. In the perfect class, processes are not suspected before they crash and in the eventually perfect, correct processes are not suspected by correct processes after a specified elapsed time. In the strong class, some processes are never suspected and eventually strong class, some correct processes are never suspected after a specified elapsed time. Sometimes, process controlled crash, where processes are given the authority to kill other processes or commit suicide, is used for failure detection [2], [42]. Fault-tolerant systems are built using a combination of algorithms failure detection using oracles, group membership service based on process-controlled crash integrated in the virtual and extended virtual synchrony, message stability using the technique of safe messages, consensus amongst member processes, resilient patterns and mechanisms for lossy channels. The failure detection, group membership service and message stability approaches seem plausible for group communication service for RAS-based high performance computing.

6 Preliminary Experience with Process Groups Algorithm

The first author's initial experience with group communication service involves designing modules as depicted in Fig.1. These object-based modules are the group membership service module, the multicast and broadcast module, the virtual and extended virtual synchrony module and the fault-tolerance and failure detection module. The group membership module includes objects for groups and processes. The message queue is treated as a referential object, supposedly to allow for easy deletion and insertion of message object. The group object employs the queue class to track process membership. Processes are allowed to join or leave the group. While the implementations of these group behaviors are preliminary, they would form the basis for future design and development of group membership service for HPC. Every process object keeps a record of its events, called the trace, in a queue fashion. No ordering mechanisms have yet been integrated; this would expectedly be included when the multicast/broadcast, virtual and extended virtual synchrony and failure detection modules have been completed.

Symmetric active replication model will be favored in RAS systems for high performance computing. This is because symmetric active replication enables fault-tolerance necessary for RAS. The model shown in the figure would support symmetric active replication by ensuring that the group communication service when successfully implemented would consist of characteristics of all four modules - effective group membership service integrated with virtual synchrony and failure detection and efficient multicast/broadcast service that would support both virtual synchrony and failure detection.

References

[1] G. V. Chockler, I. Keidar, and R. Vitenberg, Group communication specifications: A comprehensive study, *ACM Computing Surveys,* 33(4):1-43, 2001.

[2] X. Défago, A. Schiper and P. Urbán, Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey, *ACM Computing Surveys*, 36(4):372-421, 2004.

[3] C. Engelmann and S.L. Scott, High Availability for Ultra-Scale High-End Scientific Computing, in *Procs. of 2nd International Workshops on Operating Systems, Programming Environments and Management Tools for High Performance Computing on Clusters*, Cambridge, USA, June 2005.

[4] Chokchai Leangsuksun, Tong Liu, Stephen L. Scott, Richard Libby, and Ibrahim Haddad, HAOSCAR Release 1.0: Unleashing HA-Beowulf, in *Proceedings of 2nd Annual OSCAR symposium*, Winnipeg, Manitoba Canada, May 2004.

[5] Chokchai Leangsuksun, Lixin Shen, Tong Liu, Herton Song, and Stephen L. Scott, Availability Prediction and Modeling of High Availability OSCAR Cluster, in *Proceedings of IEEE International Conference on Cluster Computing* 2003, Hong Kong, December 2003, p. 380.

[6] Chockchai Leangsuksun, Tong Liu, Stephen L. Scott, and Lixin Shen, High-Availability and Performance Clusters: staging strategy, self-healing mechanism, and dependability, in *Proceedings of HPC 2003*, Sherbrooke, Canada, May 2003.

[7] Chokchai Leangsuksun, Lixin Shen, Tong Liu, Herton Song, and Stephen L. Scott, Dependability Prediction of High Availability OSCAR Cluster Server, in *Proceedings PDPTA 2003,* Las Vegas, June 2003.

[8] G. K. Thiruvathukal, Cluster Computing, *Computing in Science & Engineering*, vol. 7, no. 2, 2005, pp. 11-13.

[9] Thomas Naughton, Stephen L. Scott, Yung-Chin Fang, Phil Pfeiffer, Benoit Des Ligneris, Chokchai Leangsuksun, The OSCAR Toolkit: Current and Future Developments, *Dell Power Solutions*, November 2003, pp. 29-34.

[10] Chokchai Leangsuksun, Herton Song, and Lixin Shen, Reliability Modeling Using UML, in *Proceedings of the 2003 International Conference on Software Engineering Research and Practice*, Las Vegas, June 2003.

[11] John Mugler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, and Jeffrey M. Squyres, OSCAR Clusters, in *Proceedings of the Linux Symposium 2003*, Ottawa, Canada, June 2003.

[12] Chokchai Leangsuksun, Anand Tikotekar, Makan Pourzandi, and Ibrahim Haddad, Feasibility Study and Early Experimental Results Towards Cluster Survivability, in *Proceedings of the First International Workshop on Cluster Security, Fifth IEEE/ACM International Symposium on Cluster Computing and Grid 2005*, Cardiff, UK, May 2005.

[13] Kshitij Limaye, Box Leangsuksun, Venkata K. Munganuru, Zeno Greenwood, Stephen L. Scott, Richard Libby, Kasidit Chanchio, Grid-Aware HA-OSCAR, in *Procs of HPCS*, pp.333-339, 2005.

[14] Chokchai Leangsuksun, Tong Liu, Yudan Liu, Stephen L. Scott, Richard Libby, Ibrahim Had-dad, Highly Reliable Linux HPC Clusters: Self-Awareness Approach, *Lecture Notes in Computer Science*, Volume 3358, Nov 2004, Pages 217 -222.

[15] Ibrahim Haddad, Chokchai Leangsuksun, and Stephen L. Scott, HA-OSCAR: the Birth of High Available OSCAR, *Linux Journal*, November 2003.

[16] Idit Keidar, Jeremy Sussman, Keith Marzullo and Danny Dolev, A Client-Server Oriented Algorithm for Virtually Synchronous Group Membership in WANs, in *Procs. of the 20th International Conference on Distributed Computing (ICDCS)*, pp. 356-365, April 2000.

[17] W. Vogels and R. van Renesse, Structured Virtual Synchrony: Exploring the Bounds of Virtually Synchronous Group Communication, in *Proceedings of the 7th ACM SIGOPS European Workshop*, Connemara, Ireland, September 1996.

[18] L. Rodrigues, K. Guo, A. Sargento, R. van Renesse, B. Glade, P. Verisimo and K. Birman, A Transparent Light-Weight Group Service, in *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, pp. 130-139. Niagara-on-the-Lake, Canada, October 1996.

[19] L. Rodrigues and K. Guo, Partitionable Light-Weight Groups, in *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, April 2000.

[20] L. Rodrigues, K. Guo, P. Verissimo and K. Birman, A Dynamic Light-Weight Group Service, *Journal of Parallel and Distributed Computing*, Vol 60, pp. 1449-1479, December 2000.

[21] R. van Renesse, K. Guo, K. P. Birman, B. Glade, M. Hayden, T. Hickey, D. Malki, A. Vaysburd and W. Vogels, Horus: A Flexible Group Communications System, *Computer Science Technical Report CS-TR 95-1500, Department of Computer Science, Cornell University*, March 1995.

[22] C. Engelmann and S. L. Scott, High Availability for Ultra-Scale High-End Scientific Computing, in Proceedings of 2nd International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2), Cambridge, MA, USA, June 2005.

[23] C. Leangsuksun, V. K. Munganuru, T. Liu, S. L. Scott and C. Engelmann, Asymmetric Active-Active High Availability for High-end Computing, in Proceedings of 2nd International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2), Cambridge,

MA, USA, June 2005.

[24] C. Engelmann, S. L. Scott and G. A. Geist, High Availability through Distributed Control, in *Proceedings of High Availability and Performance Computing Workshop (HAPCW)*, Santa Fe, NM, USA, October 2004.

[25] C. Engelmann, S. L. Scott and G. A. Geist, Distributed Peer-to-Peer Control in Harness, in *Proceedings of International Conference on Computational Science (ICCS)*, Amsterdam, Netherlands, April 2002, 720-728.

[26] C. Engelmann, *Distributed Peer-to-Peer Control for Harness*, Master Thesis at the Department of Computer Science of the University of Reading, Reading, UK, Feb. 2001.

[27] D. Doley and D. Malki, The Transis approach to high availability cluster communication, *Communications of the ACM*, vol. 39(4), 1996, pp. 64-70.

[28] G. Chockler, D. Malkhi and D. Dolev, State-Machine Replication with Infinitely Many Processes: A position Paper, in *Proceedings of the International Workshop on Future Directions in Distributed Computing*, Bertinoro, Italy, 2002.

[29] K. Birman and T. Joseph, Exploiting virtual synchrony in distributed systems, in *Proceedings of the eleventh ACM Symposium on Operating Systems Principles*, 1987, pp. 123-138.

[30] Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason Hickey, Mark Hayden, Ken Birman, and Robert Constable, Building Reliable, High-Performance Communication Systems from Components, in *Proceedings of the 17th ACM Symposium on Operating System Principles* Kiawah Island Resort, SC, December 1999.

[31] K. P. Birman, A Review of Experiences with Reliable Multicast, *Software Practice and Experience*, Vol. 29, No. 9, pp, 741-774, July 1999.

[32] R. van Renesse, S. Maffeis, and K. Birman, Horus: A Flexible Group Communications System, *Communications of the ACM*, 39(4):76-83. Apr 1996.

[33] Katherine Guo, Robbert van Renesse and Werner Vogels, Structured Virtual Synchrony: Exploring the bounds of Virtual Synchronous Group Communication, in *Proceedings of the 1996 ACM SIGOPS Workshop*, Connemora, September 1996.

[34] E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony, in *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 56–65, Poznan, Poland, June 1994.

[35] Idit Keidar and Roger Khazan, A Client-Server Approach to Virtually Synchronous Group Multicast: Specifications and Algorithms, in *Procceedings of the International Conference on Distributed Computing Systems*, pp.344–355, 2000.

[36] F. B. Schneider, Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial, *ACM Computing Surveys*, 22(4), December 1990, pp.299-319.

[37] F. Cristian, H. Aghali, R. Strong and D. Dolev, Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement, in *Proc. 15th Int. Symposium on Fault-Tolerant Computing (FTCS-15)*, (Ann Arbor, MI, USA), pp.200-206, IEEE Computer Society Press, June 1985.

[38] M. Baker and R. Buyya, Cluster Computing at a Glance, in *High Performance Cluster*

Computing: Architectures and Systems, R. Buyya (eds.) Vol. 1, 1/e, *Prentice Hall*, NJ, USA, 1999.

[39] J.-M. Chang and N.F. Maxemchuk, Reliable Broadcast Protocols, *ACM Transactions on Computer Systems*, vol. 2, No. 3, pp. 251-273, 1984.

[40] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hum-mel, and H. E. Ball, An Efficient Reliable Broadcast Protocol, *Operating Systems Review*, 23(4), p.5-19, 1989.

[41] R. Baldoni, S. Cimmino, and C. Marchetti, Total Order Communications over Asynchronou Distributed Systems: Specifications and Implementations, Technical report -Dipartimento di Informatica e Sistemistica A.Ruberti, Universit´a di Roma la Sapienza -2004

[42] X. Défago, A. Schiper, and P Urbán, Comparative Performance Analysis of Ordering Strategies in Atomic Broadcast Algorithms, *IEIEC Trans. Inf. & Syst.*, 89(12), 2003.

[43] G. Morgan and P. D. Ezilchelvan, Policies for using Replica Groups and their effectiveness over the Internet, in *Proceedings of NGC 2000 on Networked group communication*, Palo Alto, California, p. 119-129, 2000.

[44] R. Guerraoui and L Rodrigues, *Introduction to Reliable Distributed Computing*, (preliminary draft), 2005.

[45] J. D. Sloan, *High Performance Linux Clusters with OSCAR, Rocks, openMosix and MPI*, O'Reilly, 2005.

[46] J. L. Schultz, Partionable Virtual Synchrony Using Extended Virtual Synchrony, Masters' Thesis, John Hopkins University, 2001.

[47] R. R. Resnick, A modern taxonomy of high availability, 1996.

[48] http://www.webopedia.com/, Jupitermedia Corporation, 2004

[49] D. Oestreicher A Simple Reliable Globally-Ordered Broadcast Service, *ACM SIGOPS Operating Systems Review*, 25(4):66-76, 1991

[50] http://en.wikipedia.org/wiki/Computer-cluster, 2004.

[51] Scientific Computing 2003 Conference High End Computing Revitalization Task Force (HECRTF) Panel Report, 2003.

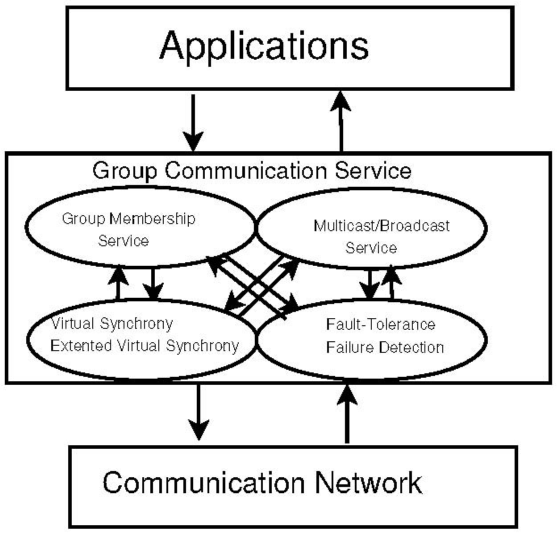

Figure 1: RAS Group Communication Service.